**/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**\* Programmer:** Razia Sultana Patan

**\* Project:** A Secured Distributed Message Delivery System using Tiny Encryption algorithm

**\* Environment:** Eclipse java developer

**\* Files Included:** PatanP1Sender.java, PatanP1Server.java, PatanP1Receiver.java, **\***  PatanP1ReceiverConnection.java, PatanP1ServerConnection.java,

**\***  userList.txt, receiverList.txt

**\* Purpose:** The authorized user can make use of Longest Common Substring (LCS) **\***  service available at receiver via relay server through encrypted

**\***  Secure channel.

**\* Input:** Messages from user inPatanP1Sender.java, list of authorized users in **\*** userList.txt, list of receivers to interact with in receiverList.txt

**\* Preconditions:** User has to type “close” message in order to shut down in any **\***  point of time.

**\* Output:** The longest common substring will be displayed to user in **\*** PatanP1Sender.java,for a request with array of substrings to the **\***  PatanP1Receiver.java via PatanP1Server.java, by authorized users.

**\* Postconditions:** Blank will be displayed, if common substring does not exist.

\***Note:** This document mainly focuses on encryption algorithm, the overall application is explained in flow chart and overall algorithm.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

**/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

\* **Overall** **Algorithm:**

\* Connect to Relay Server using port number 10630 (any)

\* Enter Username and password

\* If username and password are present in userList.txt

\* Accept the Receiver name from user

\* If receiver name is present in receiverList.txt

\* Connect to the requested receiver using port number 10631

\* Accept the count of strings to perform LCS , n

\* if the n is not numeric

\* Display message and accept the valid count number

\* Set request = n

\* for i=1 value less than or equals to n

\* Accept the string from user

\* request=count0string entered

\* Forward request=n0string10string20…0stringn to Relay Server

\* using secure channel.

\* Forward the request from Relay Server to the Receiver using

\* Secure channel.

\* Perform Longest Common Substring algorithm in Receiver

\* Forward the response = commonsubstring to Relay server

\* using secure channel.

\* Forward the same response message to Sender from Relay \* server using secure channel.

\* Display the longest common substring, its length, length of \* message to user

\* Display message, if user wants to continue, yes/no

\* Accept the input from user

\* if input= no

\* Terminate the Sender and the Receiver connection.

\* else

\* Continue steps from accepting the count to perform LCS

\* else

\* Display receiver name invalid error message

\* Accept the valid receiver name and continue step from valid \* receiver

\* else

\* Display authentication failure message

\* Accept inputs from user and continue steps from user authentication

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

**Secured Channel using Tiny Encryption Algorithm (TEA):**

**Secured Channel:**

Message

Decrypted Message

Encrypted Message

Decrypted plain text

String Decrypt\_Helper(String in)

Channel

long [] encrypt (long [])

long [] decrypt (long [])

String Encrypt\_Helper(String in)

Request in Palin Text

**Tiny Encryption Algorithm (TEA):**

|  |  |  |
| --- | --- | --- |
| **Block size:** | **32 bits** |  |
| **Number of Rounds:** | **32 cycles** |  |
| **Symmetric key size:** | **128 bits** | **4 blocks of 32 bits** |
| **Data size:** | **128 bits** | **4 blocks of 32 bits** |
| **Cypher text size:** | **128 bits** | **4 blocks of 32 bits** |
| **Padding character used, in case of variable length block:** | **1** |  |
| **Operators used per cycle** | **Bitwise shift, addition** |  |

**Encryption algorithm:**

1. Accepts input string of any size.
2. Appends “,” between every 4 characters of string (block size is 32 bits i.e. 4\*8). While forming blocks, pads with 1 for insufficient block.
3. Converts String to long using radix “36” (alpha-numeric i.e. a-z and 0-9) and stores converted input into long array.
4. Calls encrypt method for every 128 data bits i.e. for every 2 long values (2\*64 bits) and stores encrypted long bits into output long array. Uses 128-bit symmetric key for encryption.
5. Converts encrypted long values into String, appending “,” for every block of 64 bits (long block)
6. Transmits encrypted string.

**Decryption algorithm:**

1. Accepts received encryption message.
2. Splits encrypted message and converts 8 characters into long number of 64 bits using same radix “36” to preserve content.
3. Calls decrypt method for every 128 data bits i.e. for every 2 long values (2\*64 bits) and stores decrypted long bits into output long array. Uses 128-bit symmetric key and Delta sum for decryption.
4. Converts decrypted long values into String using same radix “36”.
5. Trims padded characters of 1’s and displays decrypted message.

**TEA Encryption Function:**

public long[] encrypt(long in[]) {

long[] cipher = new long[2];

int v01 = (int) in[0];

int v00 = (int) (in[0] >>> 32);

int v11 = (int) in[1];

int v10 = (int) (in[1] >>> 32);

int sum = 0;

for (int i=0; i<32; i++) {

sum += delta;

v00 += ((v01<<4) + k[0]) ^ (v01 + sum) ^ ((v01>>>5) + k[1]);

v10 += ((v11<<4) + k[0]) ^ (v11 + sum) ^ ((v11>>>5) + k[1]);

v01 += ((v00<<4) + k[2]) ^ (v00 + sum) ^ ((v00>>>5) + k[3]);

v11 += ((v10<<4) + k[2]) ^ (v10 + sum) ^ ((v10>>>5) + k[3]);

}

// In long, left part v0 and right part v1 ---v0v1

cipher[0]= (v00 & MASK32) << 32 | (v01 & MASK32);

cipher[1]= (v10 & MASK32) << 32 | (v11 & MASK32);

return cipher;

}

**TEA Decryption Function:**

public long[] decrypt(long in[]) {

long[] plainText = new long[2];

int v01 = (int) in[0];

int v00 = (int) (in[0] >>> 32);

int v11 = (int) in[1];

int v10 = (int) (in[1] >>> 32);

int sum = DECRYPT\_SUM\_INIT;

for (int i=0; i<32; i++) {

v01 -= ((v00<<4) + k[2]) ^ (v00 + sum) ^ ((v00>>>5) + k[3]);

v11 -= ((v10<<4) + k[2]) ^ (v10 + sum) ^ ((v10>>>5) + k[3]);

v00 -= ((v01<<4) + k[0]) ^ (v01 + sum) ^ ((v01>>>5) + k[1]);

v10 -= ((v11<<4) + k[0]) ^ (v11 + sum) ^ ((v11>>>5) + k[1]);

sum -= delta;

}

//long --- v0v1

plainText[0]= (v00 & MASK32) << 32 | (v01 & MASK32);

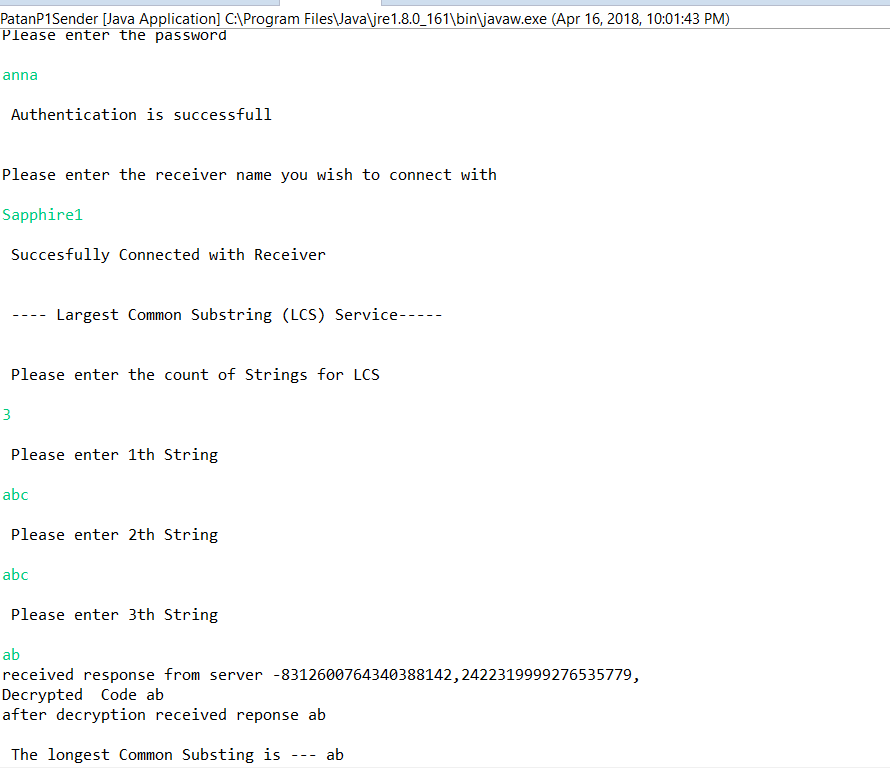
plainText[1]= (v10 & MASK32) << 32 | (v11 & MASK32);

return plainText;

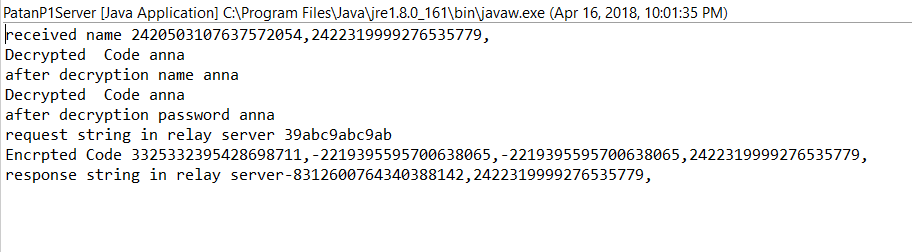
}

**Encrypted Secure Channel Implemented Screenshots:**

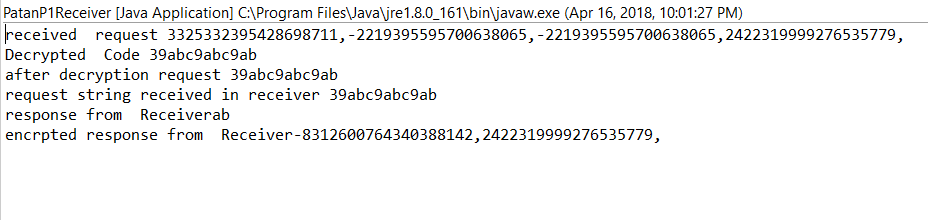
**PatanP1Sender**

****

**PatanP1Server:**

****

**PatanP1Receiver:**

****

**Flow Chart:**

**Encryption**

Input data String of any size

Chop String into block of 4 characters by appending “,” in between and pad with “1”

Convert encrypted long values into string and transmit

Encrypt 128 bits data using 128-bit symmetric key and Delta

Convert each block of string into long using radix “36” for alphanumeric values

Pass 2 long values each for encryption method (128 bits)

**Decryption**

Input Encrpyted String of any size

Chop String into block of 8 characters by splitting string using “,”

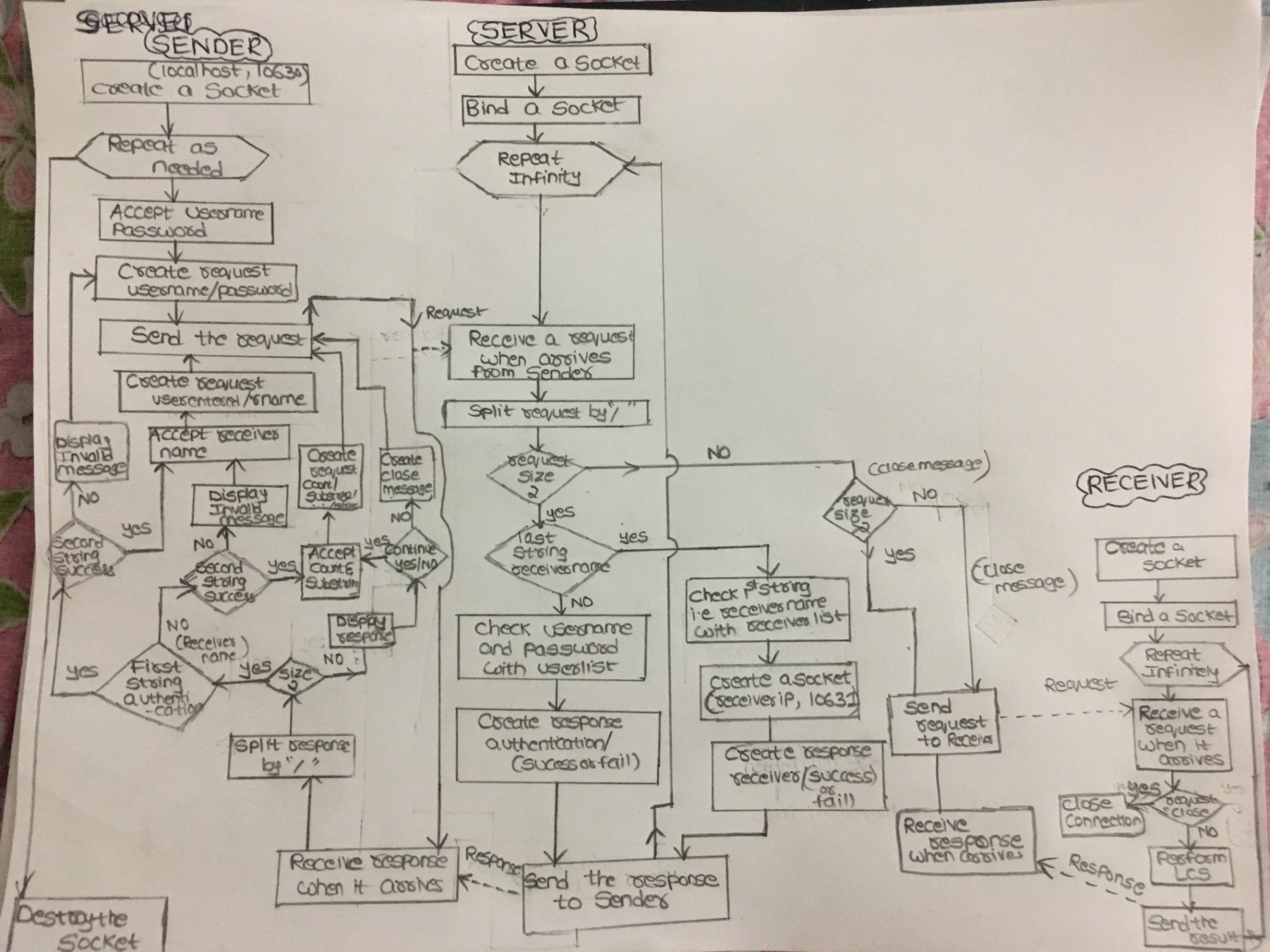
Pass 2 long values each for encryption method (128 bits)

Convert each block of string into long using radix “36” for alphanumeric values

Trim the padded 1’s and display decrypted text.

Convert decrypted long values into string

Decrypt 128 bits cipher data using 128-bit symmetric key and DeltaSum

**Overall Application Flow chart**: Before every write, we use encryption and after every read, we use decryption for secure channel communication.